

# Computer Language Interoperability Using Chasm

(or why can't I program in the language I want to?)

Craig Rasmussen

Matt Sottile

Advanced Computing Laboratory  
Los Alamos National Laboratory



## Slashdot Question

"I'm beginning to wonder if I should invest the time in learning FORTRAN. Although it is arcane, it seems to be the best tool when it comes to demanding optimization tasks and heavy computations. C/C++ does not cut it for me - it is simply too easy to make mistakes and I find myself using half of my time hunting bugs unrelated to the problem at hand. Additionally, although tools like Matlab exist they don't provide the power that justify the huge price tag they carry. I find any script based language (Matlab, Numeric Python, Scilab) to be inadequate as soon as it is necessary to use loops to describe a problem and using such tools for recursive systems can be a real pain. As another data-point, the Netlib repository seems to be very FORTRAN oriented, and it is a true gold mine when it comes to free routines for solving almost any computing task. What bothers me though is that FORTRAN code is really ugly and the language lacks almost any modern day language feature (I know about Fortran 90 but it is not much nicer than F77, and no one seems to use it). Can it really be true that the best tool we have for heavy duty computing is a 25 year old language, or have you found anything better - free or non-free?"

# Overview

- **Motivation**
- Language interoperability architecture
  - interface definition languages
  - bridging code (stubs and skeletons)
  - function argument marshalling
- Chasm example
- XML transformations (XSLT)
- Preconverted code examples

# Why Not Only FORTRAN?

- Physicists were raised on Fortran
- So why is interoperability important
  - Because there are both computer scientists and physicists
- Computer scientists prefer C++ (or Java, or Scheme, or ...)
- Students increasingly learning C++
- So why not only C++?
- "C++ is for professionals" [*Stroustrup*]

# Anecdotal Evidence

- Two scientific codes that have both Fortran and C++ versions.
- ACTS workshop, mostly Fortran with a few C++ users
- CCA Forum meeting (component middle-ware developers)
  - 1/2 person interested in Fortran
  - But Fortran required by many users of CCA
- DOE
  - Fortran, C++, and Python (Fred Johnson)
- NASA
  - Earth System Modeling Framework requires all interfaces to be in both Fortran and C++

# Fortran 95 Interoperability

- NOT!
- Fortran 95 was not designed with interoperability in mind
- We all know the problems of symbol names
  - whether to `_` or not to `_`
  - `module, symbol name example, function.in.module`
- But the real problem is with arrays
  - F95 passes arrays by descriptor
  - F95 descriptors depend on compiler vendor implementation
- No way to pass multidimensional array between C++ and F95
  - `Chasm array-descriptor library` just released on SourceForge

# Interoperability Architecture

- Motivation
- Language interoperability architecture
  - interface definition languages
  - bridging code (stubs and skeletons)
  - function argument marshalling
- Chasm example
- XML transformations (XSLT)

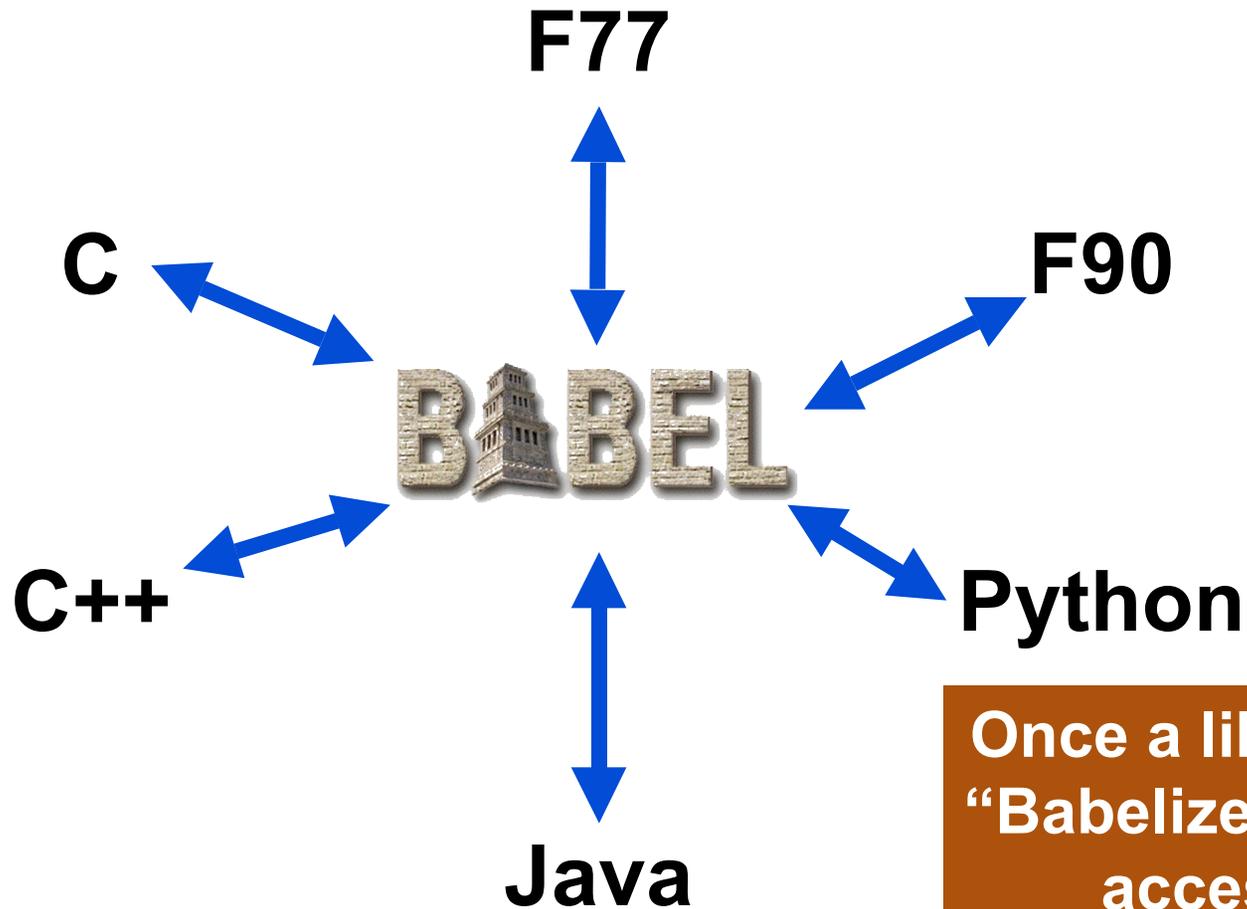
# Interface Definition Languages (IDL)

- A way to specify function interfaces in a *language independent* manner
- Corba uses an IDL
- Babel uses Scientific IDL (SIDL)
- SIDL, Function-component example:

```
version tutorial 1.0;

package tutorial {
    interface Function extends cca.Port {
        double evaluate( in double x );
    }
}
```

# Babel Make All Languages Peers



This is not  
an LCD  
Solution!

Once a library has been  
“Babelized” it is equally  
accessible from all  
supported languages

# Chasm Interfaces

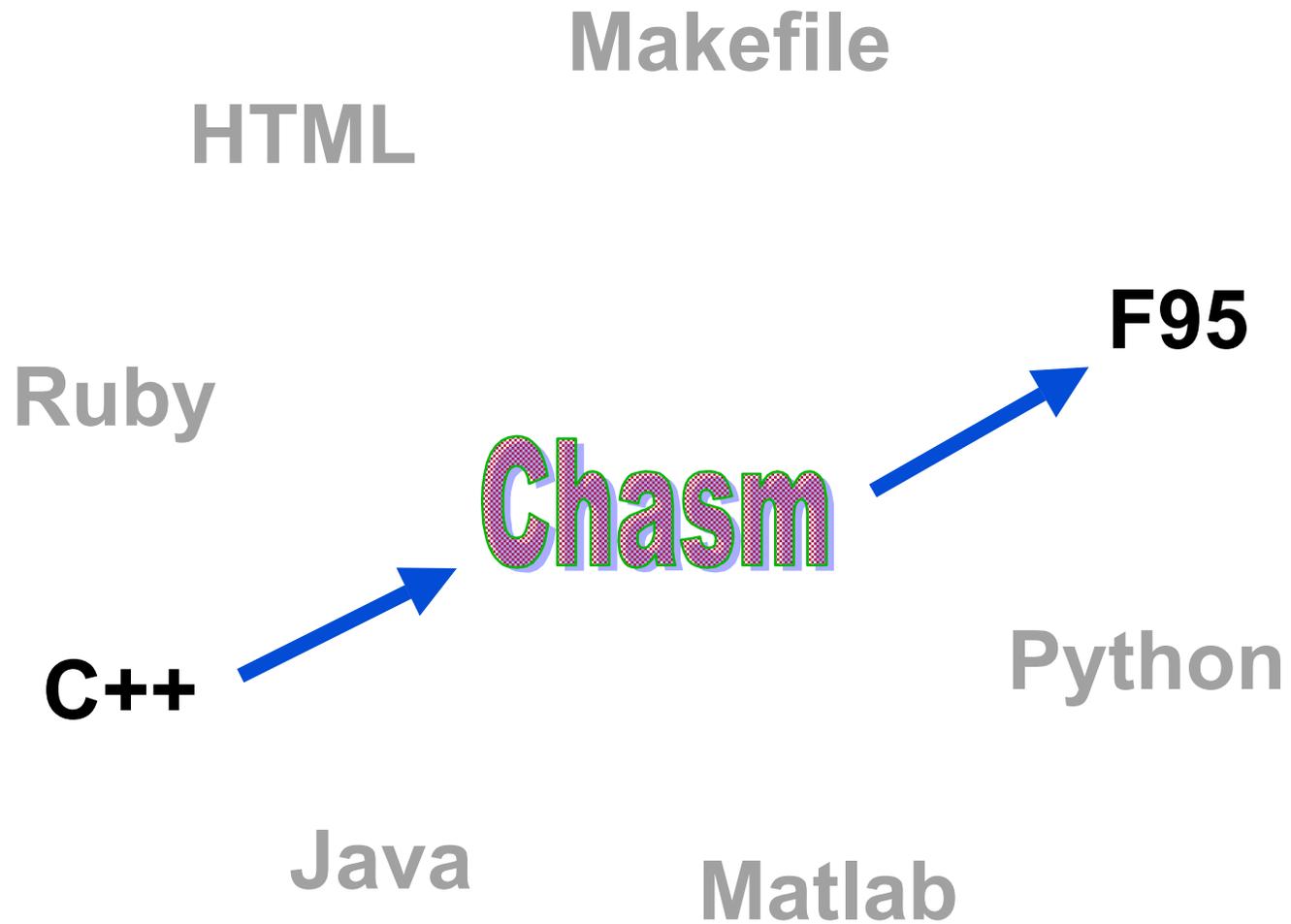
- Chasm specifies interfaces in a *language dependent* manner
  - in XML, automatically generated from user source files
- Why XML?
  - It's a standard, lots of existing tools
- Why *language dependence*?
  - use existing source files
  - potentially export more of the server language to client language
  - performance
- Example

```
<library lang="Fortran" name="LinearFunction.f90">  
  <scope name="LinearFunction">  
    <method name="LF_evaluate">  
      <arg name="x" kind="ffloat" fkind="dbl">  
      <return kind="ffloat" fkind="dbl">
```

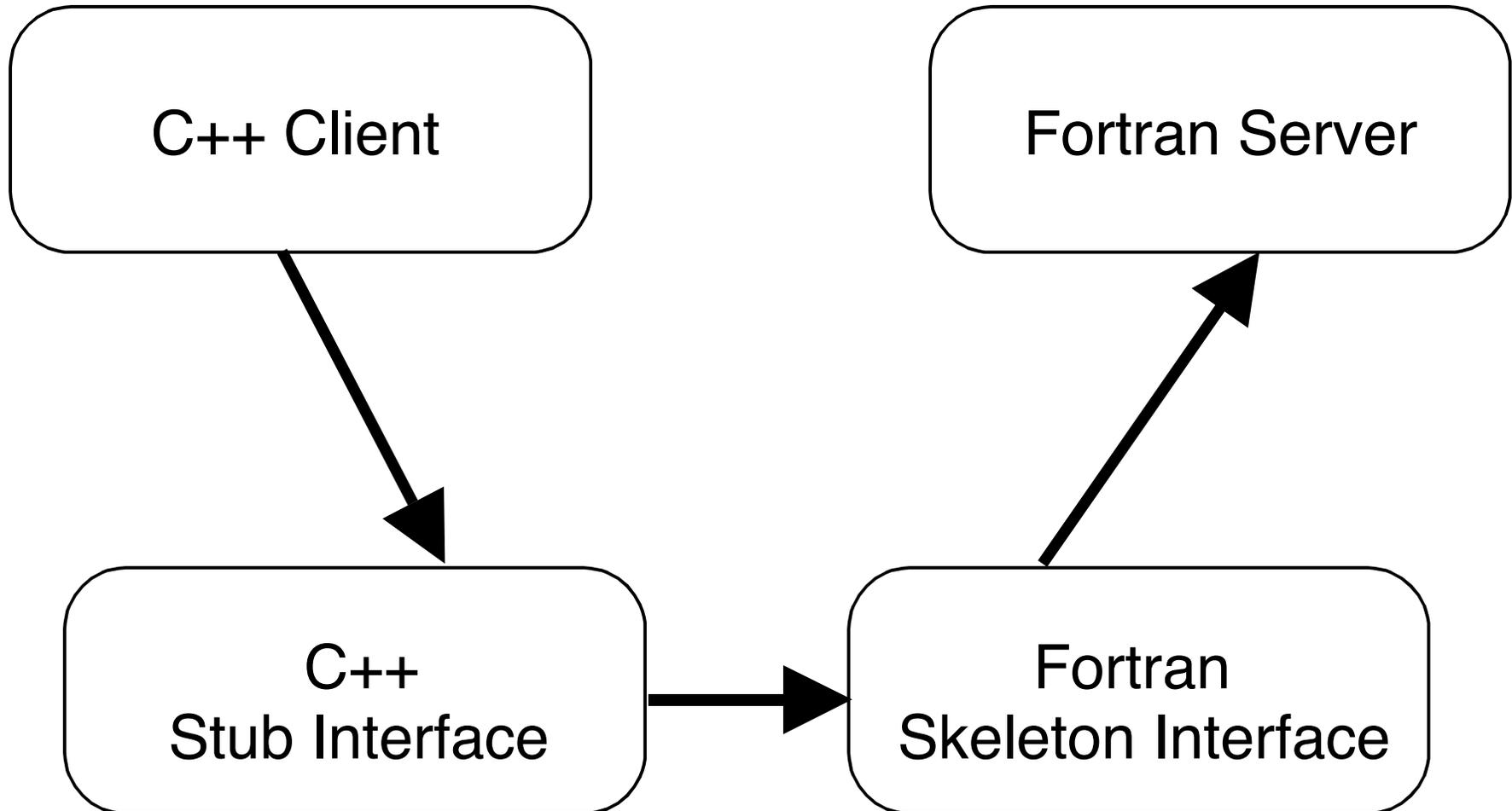
# Interoperability Architecture

- Motivation
- Language interoperability architecture
  - interface definition languages
  - bridging code (stubs and skeletons)
  - function argument marshalling
- Chasm example
- XML transformations (XSLT)

# Language Pairs



# Stub and Skeleton Code



# Stub and Skeleton Code

- Has major advantages
- Stubs
  - natural C++ interface
  - e.g., `Function::evaluate( C++ parameters... )`
  - no “\_” routines or strange macros
- Skeletons
  - marshal parameters
  - e.g., converts C++ array class to fully typed F95 array
- Chasm normally combines the two
  - reduces overhead

# Marshalling of Arguments

- Motivation
- Language interoperability architecture
  - interface definition languages
  - bridging code (stubs and skeletons)
  - function argument marshalling
- Chasm example
- XML transformations (XSLT)

# Type Conversion

F95



C++

---

integer

int &

real

float &

type(user\_type)

struct

character(len=\*)

string &

real, dimension(:, :)

F90::Array<float>

# Fortran Derived Types

- Fortran derived type is essentially just a C struct
- Chasm creates a shadow struct in C++

```
type particle
  real :: position(3)
  real :: velocity(3)
  real :: mass
end type
```

```
typedef Particle_ {
  float position[3]
  float velocity[3]
  float mass
} Particle;
```

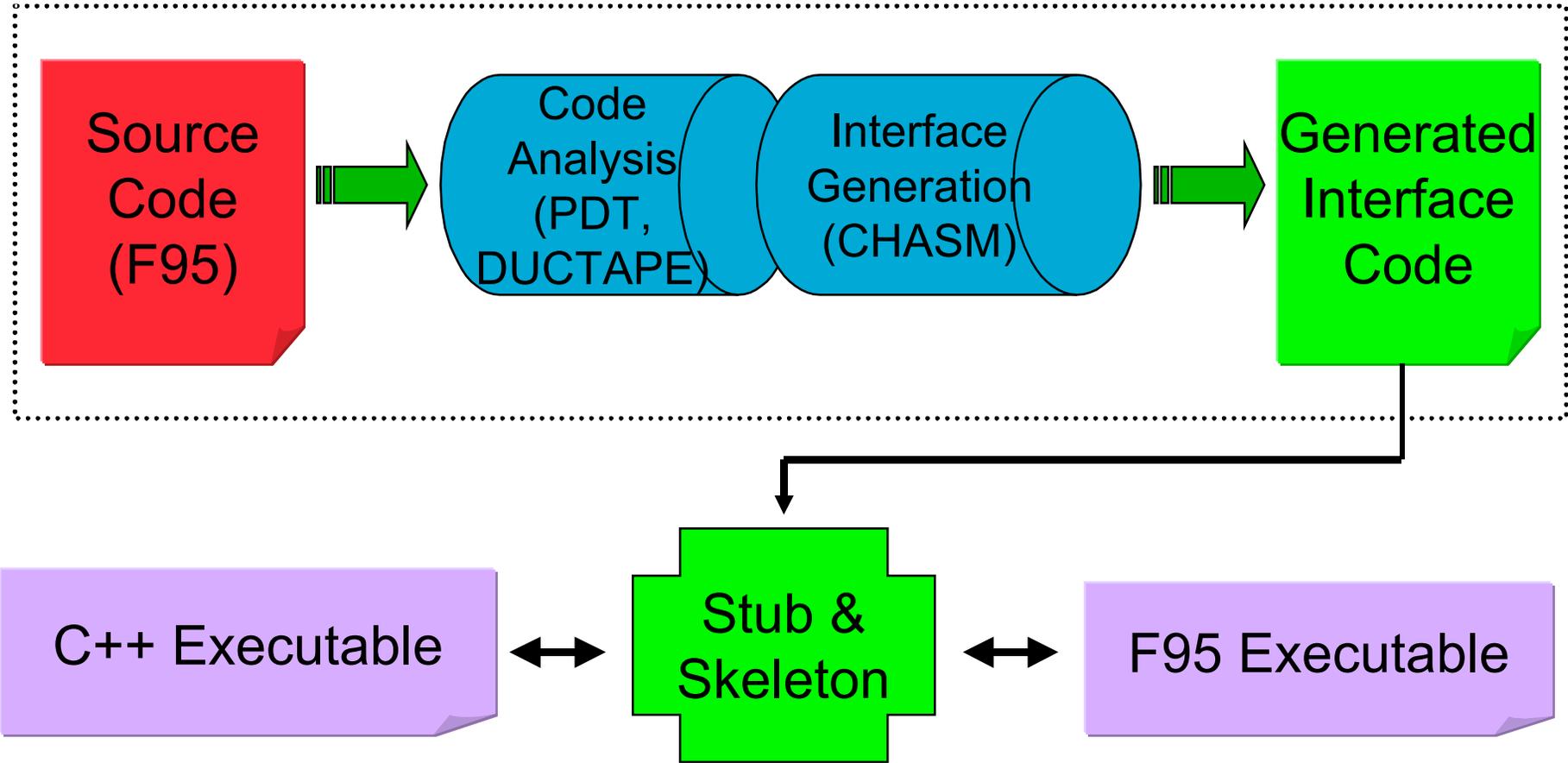
# Marshalling Fortran Arrays

- Fortran arrays passed by descriptor
  - contains array meta data, e.g.,
    - rank (# of dimensions)
    - extent (# of elements in each dimension)
- Compiler specific representation (yuk)
- F90::Array< T > class supplied
  - can supply your own
  - chasm is extensible (yea)
- C descriptor library
  - long `getArrayBaseAddress(void* desc);`

# Chasm Example

- Motivation
- Language interoperability architecture
  - interface definition languages
  - bridging code (stubs and skeletons)
  - function argument marshalling
- *Chasm example*
- XML transformations (XSLT)

# Using Chasm



# XML Transformations

- Motivation
- Language interoperability architecture
  - interface definition languages
  - bridging code (stubs and skeletons)
  - function argument marshalling
- Chasm example
- XML transformations (XSLT)

# Stylesheet Example

- Code segment to print the names of procedures in a module

```
<library name="LinearFunction.f90" lang="f90">
  <scope name="LinearFunction">
    <method name="Evaluate_Linear" kind="static">
      <arg name="X" kind="ffloat" fkind="dbl"/>
      <return kind="ffloat" fkind="dbl"/>
    </method>
  </scope>
</library>
```

```
<xsl:template match="/">
  <xsl:apply-templates select="library/scope/method"/>
</xsl:template>

<xsl:template match="library/scope/method">
  <xsl:text> Found procedure: </xsl:text>
  <xsl:value-of select="@name"/>
  <xsl:value-of select="$newline"/>
</xsl:template>
```

# Contributed Code

- Chasm distribution has area for contributed code
- Sometimes we put things there
  - bridging code already generated
  - complete with makefiles
- Will show two examples soon to be added
  - Selected access to C++ STL library
  - Selected access to C++ VTK graphics library

# Access to C++ STL

- Example, Fortran associative map
- Store a piece of information with an associated name
- Retrieve it later by its name

```
use STL_Map
type(STL_Map_t) :: imap

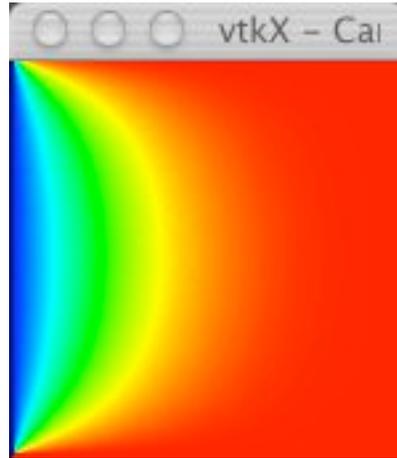
call new_STL_Map(imap)

call insert(imap, "one", 1)
call find(imap, "one", i)

call destroy(imap)
```

# Access to C++ VTK graphics

- Color graphics



```
use vtkRenderWindow
type(vtkRenderWindow_t) :: visRenderWindow

visRenderWindow = new_vtkRenderWindow()
call SetSize(visRenderWindow, width, height)
!
! call CopyImportVoidPointer(imageImport, array,
!                               elemSize*width*height)
!
call Render(RenderWindow)
```

# Chasm Status

- C array-descriptor library completed for most compilers (Absoft, Compaq, IBM, Intel, Lahey, NAG, MIPSpro, Sun)
  - Portland compiler (PGI) will be supported by vendor
  - Cray coming soon
- C++ calling Fortran mostly completed
- Fortran calling C++ work begun
- Demonstration stylesheet transformations for
  - Fortran CCA Components
  - Ruby
- Would like to talk to people interested in
  - Matlab
  - Python
  - ?

# Conclusions

- Chasm transforms code
- Why use Chasm
  - shear drudgery of hand coding
  - bridging code doesn't get out of sync
  - impose project-wide language transformation standards
- Chasm is extensible
  - modify code generation to fit the needs of your project
- No long-term "dependency" on Chasm project
  - except for array-descriptor library?
- Your project "owns" the generated code

## Links

[Babel](http://www.llnl.gov/CASC/components/) - <http://www.llnl.gov/CASC/components/>

[Chasm](http://sourceforge.net/projects/chasm-interop/) - <http://sourceforge.net/projects/chasm-interop/>